

# Current Trends in High-Level Synthesis of Asynchronous Circuits

**Sparsø, Jens**

*Published in:*
16th IEEE International Conference on Electronics Circuits and Systems (ICECS)

*Link to article, DOI:*
10.1109/ICECS.2009.5411011

*Publication date:*
2009

*Document Version*
Publisher's PDF, also known as Version of record

# Current Trends in High-Level Synthesis of Asynchronous Circuits

Jens Sparsø
Department of Informatics and Mathematical Modelling, Technical University of Denmark
Richard Petersens Plads, Building 322, DK-2800 Kgs. Lyngby, Denmark
Email: jsp@imm.dtu.dk

*Abstract*— **This paper is a survey paper presenting what the author sees as two major and promising trends in the current research in CAD-tools and design-methods for asynchronous circuits. One branch of research builds on top of existing asynchronous CAD-tools that perform syntax directed translation, e.g. the Haste/TiDE tool from Handshake Solutions or the Balsa tool from the University of Manchester. The aims are to add high-level synthesis capabilities to these tools and to extend the tools such that a wider range of (higher speed) micro-architectures can be generated. Another branch of research takes a conventional synchronous circuit as the starting point, and then adds some form of handshake-based flow-control. One approach keeps the global clock and implements discrete-time asynchronous operation. Another approach substitutes the clocked registers by asynchronous handshake-registers, thus creating truly continuous-time asynchronous circuits that operate without a clock. The perspective here is that the substitution/conversion is done as the final step in an otherwise conventional synchronous design flow.**

## I. Introduction

Asynchronous circuits use local handshaking to control the transfer of data between components (down to the level of combinatorial blocks and registers). This gives asynchronous circuits a range of characteristic features which can be exploited in different contexts to obtain one or more of the following properties: high speed, low power consumption, easy implementation of (dynamic) voltage scaling and a high degree of modularity. Furthermore, in the SIA ITRS 2007 [1] asynchronous circuit techniques are listed among the measures which can be used to cope with parameter uncertainty and timing variability. The use of asynchronous circuits tackles the effects of correlated variability sources, such as supply voltage, operating temperature, and large-scale process variations, and provides solutions to system-wide interconnect.

Asynchronous circuit techniques have been employed in products for many years, and a number of current start-up companies are providing a range of chips, IP-cores and interconnect fabrics [2], [3], [4], [5], [6]. In most cases, when using these predesigned components and chips, a system designer does not need to know about the underlying asynchronous design techniques.

In order to enable more widespread use of asynchronous design, efficient and easy to use high-level CAD-tools are required. Over the years many different researchers, primarily at universities, have addressed this problem and many fundamental problems have been solved. But few groups have had the resources necessary to bring the work beyond what can be described as demonstration prototypes or elements which could be integrated in a complete design flow. In order to bring research closer to practical use, it seems natural and necessary to tie into some existing design flow. This paper reviews two such efforts.

So far the most commonly used and most powerful tools for synthesis of large-scale asynchronous circuits are based on a technique known as *syntax directed translation*. The Haste/TiDE tool from Handshake solutions [5] and the Balsa tool [7], [8] from the University of Manchester are two well known examples. These tools perform a one-to-one mapping of the syntax-tree of the source program, into a corresponding structure of handshake components and this essentially means that no optimizations are performed. Furthermore the circuits operate in a control driven manner, and this tends to limit the speed. Several research groups are currently addressing these deficiencies as explained in section II.

Another and very different class of research, aims at converting a clocked circuit – synthesized using conventional CAD-tools – into an equivalent asynchronous circuit, by employing a set of transformations and component substitutions as explained in section III. While this may not enable a designer to fully exploit the potential of asynchronous design, it may be what is needed to obtain circuits which better tolerate variability, and which suffers less from dynamic voltage (IR) drops and power grid noise caused by synchronous clocking.

It should be emphasized that the four-page format does not allow a comprehensive coverage of all current research – many more approaches than the above two are being pursued including [9], [10]. While these often involve the development of new tool-flows, the approaches presented in this paper deliberately aim at supplementing and building on top of existing and mature design tools. Finally a pointer to a recent and quite extensive survey of asynchronous design flows that can tackle large designs seems appropriate [11].

## II. Extending syntax-directed-translation

As mentioned above, the most commonly used and most powerful tools for synthesis of large-scale asynchronous circuits are based on a technique known as ”syntax directed translation”; a process in which a description in a CSP-like language [12] is mapped directly into a hardware implementation composed of so-called handshake components. Using conventional EDA-tools for technology mapping, a

978-1-4244-5091-6/09/$25.00 ©2009 IEEE

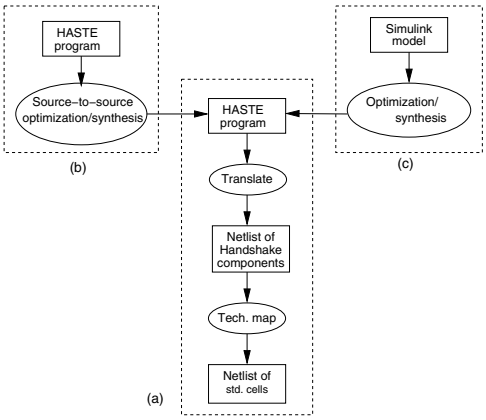


Fig. 1. (a) Handshake Solutions' Haste/TiDE design flow. (b-c) Several possible behavioural/high-level synthesis front-end tools for the Haste/TiDE design flow.

standard cell netlist is then produced. One such tool which have recently been made commercially available is the Haste design language and the associated TiDE design environment from Handshake Solutions [5]. The tool was formerly known under the name Tangram [13], [14]. Figure 1(a) shows the Haste/TiDE design flow. The translation performed by the Haste/TiDE tools is essentially a one-to-one mapping of the syntax-tree of the source Haste program, into a corresponding structure of handshake components. This transparency is both an advantage and a disadvantage. The disadvantage is that in order to explore alternative implementations, the designer is required to actually program these. The advantage is that the designer has full control over the resulting circuit. And as we will see in the following subsections, this can exploited by automatic synthesis and optimization tools, as it allows precise specification of an implementation at a high level.

### A. Haste-to-Haste optimizations

Research at the Technical University of Denmark [15], [16], [17] has resulted in the development of a behavioural synthesis frontend to the Haste/TiDE tool as illustrated in figure 1(b). Input to the tool is a behavioural description in the Haste language, and output from the tool is a Haste program describing the synthesized implementation consisting of a datapath and a controller. Figure 2 shows a generic example of such an implementation template using handshake components.

The synthesis tool performs the following sequence of operations: (i) extraction of a control data flow graph (CDFG) representation of the source Haste-program, (ii) synthesis of an implementation which meets the required constraint, and (iii) generation of a Haste-program describing the synthesized implementation. The synthesis step solves the classic problems of resource sharing, scheduling and binding. Although the scheduling is done using a discrete-time model (as in synchronous design), it should be stressed that the implementation illustrated in figure 2 explicitly implements the actual dependencies among the operations performed. The tool has

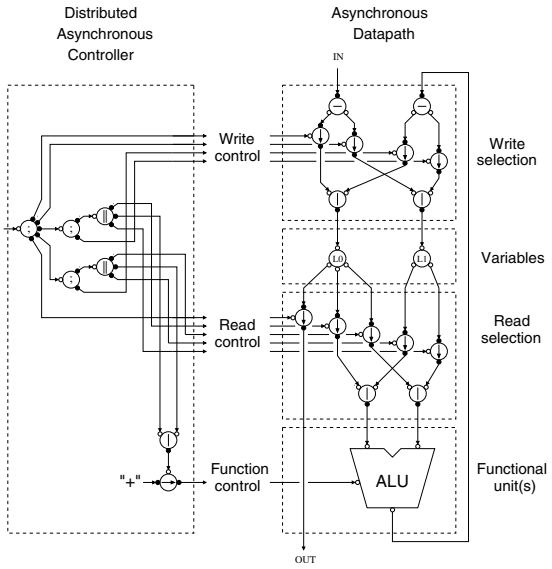


Fig. 2. Implementation of a datapath and its associated controller using handshake components.

been used to optimize a number of well known benchmarks The results show that it is possible to achieve an average $30\%$ area reduction when optimizing for area, and an average $40\%$ increase in speed when optimizing for speed.

Research at the University of North Carolina at Chapel Hill considers a similar Haste-in Haste-out front-end, figure 1(b), but with the aim of optimizing the speed of the circuit [18]. This is done using techniques like loop unrolling and pipelining.

It is interesting to note that the two approaches described above complement each other: A designer using the Haste/TiDE design flow may use the tool presented in [18] in order to automatically optimize for speed, and he may use the tool presented in [17] to optimize for area, and more generally, for constraint driven design space exploration. Finally, it seems to this author, that this kind of source-to-source optimization/transformation using syntax-directed-translation tools like Haste/TiDE represents a promising direction for further research.

### B. A Simulink to Haste front-end

Some designers may find the Haste language 'unconventional', and to remove this barrier the use of more well known and widely used languages must be considered. Furthermore there is a continuing evolution towards specifying designs at higher levels of abstraction. One example is tools which are capable of synthesizing circuit implementations directly form MATLAB/Simulink models. Such tools exist for synchronous design, and as Simulink is based on a data-flow model of computation, one could expect that asynchronous synthesis tools could be developed as well. In [19] researchers at Politecnico di Torino and Handshake Solutions have explored this and outlined an approach for automatically generating Haste code from Simulink specifications, figure 1(c).

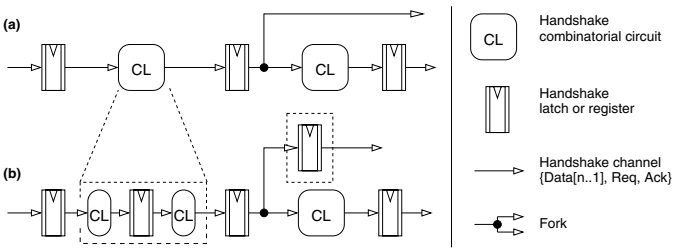


Fig. 3. (a) An example asynchronous circuit. (b) The same circuit to which some pipeline registers has been added.

### C. Data-driven circuits

While the Haste/TiDE and Balsa tools have been used to produce circuits with interesting properties including low power and inherent adaptation to supply voltage scaling, it is widely accepted that it is difficult to design high-speed circuits using these tools. The problem is that the synthesized circuits are control driven. Both input and output ports on variables (i.e., registers and or latches) are passive as indicated by the unfilled circles on the ports of the variables in figure 2. Components which are called transferrers, and which are controlled by yet other handshake circuits are needed to actively pull (i.e., read) data out of the passive variable and to actively push (i.e., write) data into the passive variables. While this tends to limit the amount of data transfers to a minimum (thus saving power), it also adds considerable latency. In order to enable higher speed it is necessary to support a data-driven pipelined design style as well. Research at the University of Manchester [20] has addressed this by extending the Balsa language with constructs that will allow a designer to express data-flow. This must be supported by a set of data-flow handshake-components such as combinatorial circuit blocks and variables, both with passive input ports and active output ports, such that control-less data-driven pipelined circuits may be generated.

## III. Transforming synchronous designs

Asynchronous circuits are based on a token flow model of computation [21] and most asynchronous circuits retain their function if handshake latches/registers are added to the circuit. What matters is the flow of data-tokens in the circuit. This gives asynchronous circuits an elasticity which is not found in simple synchronous circuits, where the addition of a clocked register in some signal path will mess up the computation performed by the circuit. Figure 3(a) shows an example asynchronous circuit and figure 3(b) shows the same circuit, which has been pipelined to gain additional speed. The circuits in figure 3 exhibit the same functionality.

An interesting and promising body of research aims at adding similar token-flow based elasticity to a clocked design and eventually transforming a synchronous design into a corresponding asynchronous one. The perspective is obvious; a circuit can be designed using any existing synchronous CAD-tool and finally transformed into an equivalent asynchronous one.

### A. Latency insensitive design

In deep sub-micron CMOS-technologies the latency associated with wires is considerable and may not be known until a post-layout simulation can be made, and long wires may have latencies of several clock cycles. In response to challenges such as these *latency insensitive design* [22] has been proposed. The circuits are clocked and can be designed using existing CAD tools, but they use request-acknowledge based protocols and are designed such that addition of registers does not change the functionality of the circuit. This ability to break long combinatorial paths (logic as well as wires) makes the design more modular. In [22] it is explained how this enables a separation of computation and communication, and how this allows a change from computation-bound to communication-bound design, thereby offering a solution to the design of complex systems designed by integrating many predesigned components (so called IP-based systems-on-chip). A design methodology is presented, where IP-blocks are extended with buffered communication channels on their ports, and where buffered channel repeaters (called relay stations) can be inserted.

### B. Synchronous elastic systems

The same ideas have been pursued and refined further in [23] where the term *synchronous elastic* is used to denote the architectures and handshake protocols. Synchronous elastic systems may be seen as time-discrete asynchronous systems, and they enjoy the same properties as fully asynchronous systems . As stated in the abstract for a tutorial presented at ASYNC'08 [24]: *"Such systems are 'time elastic' in a sense that they can tolerate dynamic and static changes in latencies of computation and communication components. Therefore, they enable new micro-architectural trade-offs, e.g. a wider use of variable latency components targeting average case optimization, rather than the worst case optimization traditional to regular synchronous circuits. They also enable correct-by-construction re-pipelining of wires and computation blocks – a useful feature that can simplify design and RC scaling in the nano-scale technologies. In comparison with continuous time asynchronous systems (operating without a clock) synchronous elastic have a few advantages: complete reuse of the synchronous CAD tools and design practices and negligible overhead in area and delay (if constructed using an efficient technique)"*.

Having said this, it is still necessary to distribute a global clock, and to solve the associated buffering and skew problems.

### C. Synchronous handshake components

As an aside it is interesting to note that a synchronous back-end exists for the Haste/TiDE tool described earlier. This back-end maps a circuit described in Haste into a net-list of *clocked* handshake components [25]. The resulting circuits are similar to the above mentioned latency insensitive and synchronous elastic systems, but there is also one important difference: they are control driven (using passive variables) rather than data

driven. As explained previously, this may impact (i.e., reduce) speed and power consumption.

### D. De-synchronization

The de-synchronization method described in [26] may be seen as taking the above ideas one step further in the sense that a synchronous circuit is transformed into a truly asynchronous circuit by substitution of clocked registers by asynchronous handshake-registers. The synchronous circuit is a pure clocked circuit without any of the flow-control or handshaking described above. Therefore, a key issue in this work is to ensure that the functionality of the original circuit is maintained. A notion of (data-token) flow equivalence is introduced and used to formally prove that the transformation preserves the functionality. De-synchronization has been used to implement an asynchronous DLX-processor [27] and an effort to further develop and commercialize the technology is underway [28].

## IV. Discussion and conclusion

The paper presented what the author sees as two major and important directions of research aiming at providing CAD-tools and design methods for asynchronous circuits.

One line of research builds on top of existing stable and state-of-the-art CAD tools (Haste/TiDE and Balsa). The research efforts described in this paper seek to extend the tools towards higher levels of design. The fact that an intended implementation can be expressed (at a high level) using the Haste or Balsa languages, represents an important and interesting advantage/simplification when developing such high level tools. The paper reviewed efforts addressing behavioural synthesis, synthesis from modeling languages like Simulink, automatic structural optimizations and language features supporting the design of high-speed data driven circuits. More research in this direction is envisioned and encouraged.

Another line of research aims at: (i) adding asynchronous concepts to the synchronous design domain and/or (ii) transforming synchronous circuits into equivalent asynchronous ones. The synchronous elastic and the latency insensitive approaches are examples of the former and effectively involve the design of discrete-time (i.e. clocked) asynchronous circuits. As the circuits are designed to explicitly implement the intended token flow, the author would expect that these circuits could be transformed directly into fully asynchronous ones. Finally de-synchronization is a technique which transforms a conventional synchronous circuit into a fully asynchronous one. In this process (token) flow equivalence is an important issue. The perspectives here are that the sharp distinction between synchronous and asynchronous design is fading, and that conventional synchronous CAD-tools can be used to design synchronous circuits, which are subsequently transformed into asynchronous ones in a final step of the design flow.

## References


[1] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors," http://www.itrs.net/home.html, 2007.

[2] Achronix Inc., http://www.achronix.com.

[3] Tiempo Inc., http://www.tiempo-ic.com.

[4] Fulcrum Microsystems Inc., http://www.fulcrummicro.com.

[5] Handshake Solutions Inc., http://www.handshakesolutions.com.

[6] Silistix Inc., http://www.silistix.com.

[7] D. Edwards and A. Bardsley, "Balsa – an asynchronous hardware synthesis system," in *Principles of asynchronous circuit design – A systems perspective*, J. Sparsø and S. Furber, Eds. Kluwer Academic Publishers, 2001, ch. 9–12, pp. 155–218.

[8] ——, "Balsa: An Asynchronous Hardware Synthesis Language," *The Computer Journal*, vol. 45, no. 1, pp. 12–18, 2002.

[9] D. Shang, F. Burns, A. Koelmans, A. Yakovlev, and F. Xia, "Asynchronous system synthesis based on direct mapping using VHDL and Petri nets," *IEE Proc., Comput. Digit. Tech.*, vol. 151, no. 3, pp. 209–220, May 2004.

[10] G. Venkataramani, M. Budiu, T. Chelcea, and S. C. Goldstein, "C to asynchronous dataflow circuits: An end-to-end toolflow," in *International Workshop on Logic synthesis (IWLS)*, Temecula, CA, June 2004, pp. 501–508.

[11] A. Taubin, J. Cortadella, L. Lavagno, A. Kondratyev, and A. Peeters, "Design automation of real life asynchronous devices and systems," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 1, pp. 1–133, 2007.

[12] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, Aug. 1978.

[13] C. Niessen, C. van Berkel, M. Rem, and R. Saeijs, "VLSI programming and silicon compilation," in *Proc. International Conf. Computer Design (ICCD)*. IEEE Computer Society Press, 1988, pp. 150–166.

[14] C. H. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, "The VLSI-programming language Tangram and its translation into handshake circuits," in *Proc. European Conference on Design Automation (EDAC)*, 1991, pp. 384–389.

[15] S. F. Nielsen, "Behavioral synthesis of asynchronous circuits," Ph.D. dissertation, Technical University of Denmark, Dept. of Informatics and Mathematical Modelling, 2005, iMM-PHD-2005-144.

[16] S. F. Nielsen, J. Sparsø, and J. Madsen, "High-level synthesis of asynchronous circuits using syntax directed translation as backend," *IEEE Transactions on VLSI Systems*, vol. 17, no. 2, pp. 248–261, 2009.

[17] S. F. Nielsen, J. Sparsø, J. Jensen, and J. Nielsen, "A Behavioral Synthesis Frontend to the Haste/TiDE design flow," in *Proc. International Symposium on Asynchronous Circuits and Systems*. IEEE Computer Society Press, 2009, pp. 185–194, (Best paper finalist).

[18] J. Hansen and M. Singh, "Concurrency-enhancing transformations for asynchronous behavioral specifications: A data-driven approach," in *Proc. IEEE International Symposium on Asynchronous Circuits and Systems*. IEEE Computer Society Press, 2008, pp. 15–25.

[19] M. Tranchero, L. M. Reyneri, A. Bink, and M. de Wit, "An automatic approach to generate haste code from simulink specifications," in *15th IEEE International Symposium on Asynchronous Circuits and Systems*. IEEE Computer Society Press, 2009, pp. 175–184.

[20] S. Taylor, D. Edwards, and L. Plana, "Automatic compilation of data-driven circuits," in *14th IEEE International Symposium on Asynchronous Circuits and Systems*. IEEE, 2008, pp. 3–14.

[21] J. Sparsø and S. Furber, Eds., *Principles of asynchronous circuit design – A systems perspective*. Kluwer Academic Publishers, 2001.

[22] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Coping with latency in SOC design," *IEEE Micro*, vol. 22, no. 5, pp. 24–35, 2002.

[23] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *2006 43rd ACM/IEEE Design Automation Conference*. IEEE, 2006, pp. 657–662.

[24] Mike Kishinevsky, "Synchronous Elastic Systems," http://async.org.uk/async2008/keynote-tutorials.html, 2008, Tutorial presented at the ASYNC'08/NOCS'08 in Newcsstle, UK.

[25] A. M. G. Peeters and K. van Berkel, "Synchronous handshake circuits," in *ASYNC*, 2001, pp. 86–95.

[26] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications coping with latency in soc design," *IEEE Transactions on Computer-Aided Design*, vol. 25, no. 10, p. 19041921, 2006.

[27] M. Amde, I. Blunno, and C. P. Sotiriou, "Automating the design of an asynchronous DLX microprocessor," in *DAC '03: Proceedings of the 40th annual Design Automation Conference*. ACM, 2003, pp. 502–507.

[28] Elastix Inc., http://www.elastix-corp.com/.